

Subscribe to DeepL Pro to edit this document.  
Visit [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document) for more information.

Computer Vision Final Project: Report on the implementation of an edge detector

2017103580 Jungwoon Kim, Department of Sociology

1.Analyze the problems and limitations of existing algorithms

The process of finding edges is divided into two parts: first, we reduce the noise in the image and then find the brightness difference with the neighboring pixels. If the value is greater than a certain threshold, the pixel is declared as an edge. Specifically, box-filter and Gaussian-filter are used to remove noise from an image, but both methods assume a specific model, which means that they are not flexible and may not perform well if the color or brightness of the image to be processed changes rapidly. It's also not easy to set the right size (or variance) for both filters. The problem with the latter in particular is that if the sample size (or variance) used by the filter is too large, the computation will include pixels with different parameters, resulting in bias, and if it is too small, the central limit theorem cannot be satisfied. This preprocessing (especially box-filter) also has the disadvantage of distorting the data, resulting in thicker edges, which requires additional computations (such as canny edge detection) to fix.

The Gaussian edge detector also performs an exponential computation for each pixel, which for very large M requires a computation of for a very large M for each pixel, which increases the computational complexity of the algorithm. In addition, existing algorithms judge edges by the amount of brightness change, which has limited expressive power, i.e., if the sum of RGB values is the same despite different colors, it may not be classified as an edge because the brightness difference is small. Finally, there is the issue of setting the threshold, which can be subjective.

Therefore, in this report, we will present an edge detector algorithm that can solve the above problem. In a nutshell, the information we use to determine the edge is not only the traditional brightness, but also RGB or brightness+RGB. We also classify each pixel into a specific group, and then refer to its neighbors (up and down or left and right). If the pixel and its neighbors are in different groups, we classify the pixel as an edge, and if you want, we also check whether the pixel is noisy and factor that into the edge classification. This classification is done for every pixel like a clustering algorithm, so there is more information to use, and it is also more flexible because it does not compare values only in the local area.

2.Explain the proposed algorithm

Reference 1 below shows the pseudocode for the image preprocessing method, and Reference 2 is the algorithm to determine whether a particular pixel is an edge or not. Image preprocessing can be categorized into three types depending on how the similarities are grouped, but the common thread is that even if the value of a particular pixel is deterministic or a random variable, the variance is very small. This is related to the fact that when classifying pixels into a group, the similarity is determined by comparing specific pixels in the group to the object to be classified, rather than using the average of the group as in other clustering algorithms. If the variance of the pixels is large, this approach challenges the representativeness of the pixels, making the computation unreliable. Therefore, the value of a particular pixel must be constant in order to use this algorithm, and to check this, you can take the same photo several times and compare the RGB values.

The information available for image preprocessing is brightness, color, and brightness+color. First of all, the brightness uses the average value of RGB as in the traditional algorithm, but it utilizes Euclidean distance to measure similarity. If the difference between the brightness of a pixel in a certain group and the other pixels is less than a threshold, the latter are classified into that group; otherwise, the object is classified into a new group. The representative value of the created group is the information of the first classified pixel. In addition, the threshold for setting the optimal edge varies from image to image, and to determine it statistically, the difference between the brightness of each pixel and its average value is found. The threshold is then selected as the value that is significantly away from the average of the values. The threshold is determined in the same way as a conventional statistical test, and if the difference between the brightness of a particular pixel and other brightnesses in the brightness difference distribution is significantly large, it can be categorized into different groups.

Here's how to do this Find the difference (absolute value) between each pixel and the average of the entire image and sort them in ascending order. We'll take two p-values with a 5% significance level and the median as our threshold. This way, the threshold can be determined from the data, removing the subjectivity of the user. And a larger threshold is more likely to cluster those pixels into a specific group, which is equivalent to increasing the size of the box-filter (or increasing the variance of the Gaussian). A value that is significantly large from the mean of the brightness difference can be used to get a significant edge, and a value that is small from the mean can be used to judge noise because the threshold is small. The reason for providing the median is to support the possibility of using multiple thresholds to obtain edges in the future.

The way we classify certain pixels into groups by color is by using intrinsics in RGB space. If the color vectors of two objects are in the same direction in RGB space, their intrinsic is equal to 1, which means that their colors are similar. Also, since the RGB space can only take positive numbers at all times, the minimum value of the inner product is zero. So we find the inner product between the vector representing a particular group and the RGB of a pixel, and if it is greater than the threshold, we classify the pixel as belonging to that group. Otherwise, we classify the object into a new group and declare its value as the representative value of the created group. Similarly, the way to use color and brightness simultaneously is to find the inner product of objects and groups in a four-dimensional space (RGB+brightness), which is the original RGB space plus brightness, and compare it to a threshold. Since these inner operations can be represented as matrices, they can be computed on the graphics card to speed things up.

<Reference 1: Pseudocode for how to preprocess images

|  |
| --- |
| def pre\_processing(Image, threshold):  result =deepcopy(Image) A two-dimensional list, the size of which is equal to the input image.  group=[image[0][0]] #Need to set an initial value to categorize the pixels into groups.  result[0][0]=[group[0], 0] #Store the pixel's information (RGB or brightness) and the group index of the pixel.  while (have we classified all the pixels in the image):  similarity=min(group-Image[i][j]) #Find the group with the highest similarity to the object and save the similarity.  group\_index=argmin(group-Image[i][j]) #store the index of the group with the highest similarity.  if(similarity>threshold): #Classify the pixel into the corresponding group if the similarity is higher than the threshold.  result[i][j]=[Image[i][j], group\_index]  else: #If the similarity is lower than the threshold, classify the pixels into a new group and take them as representative of the value.  group.add(image[i][j]); new\_group\_index=group index of pixels categorized into the new group  result[i][j]=[Image[i][j], new\_group\_index]  return result |

<Reference 2: Pseudocode for how to determine whether a pixel is an edge or not>.

|  |
| --- |
| def get\_edges(pre\_processing(Image, threshold):  result=deepcopy(Image) #return a value whose size is equal to Image.    while(checked for edge for all pixels in the image):]  # if the grouping of a particular pixel is different between left and right or top and bottom pixels, classify that pixel as an edge  edges\_test1=(result[i][j][1]!= result[i][j+1][1]) or result[i][j][1]!= result[i][j-1][1])) #left-right comparison  edges\_test2=(result[i][j][1]!= result[i-1][j][1]) or result[i][j][1]!= result[i+1][j][1])) #Comparison up/down  if(edges\_test1 or edges\_test2):  result[i][j]=1  else:  result[i][j]=0  return result |

This categorization of each pixel into a specific group is a process of reducing noise in the data, like a box or gaussian filter, because to categorize into a group is to replace the RGB or brightness of that pixel with the value of the group. Since this method does not average the pixels like box and gaussian, it cannot use the central limit theorem, but since we do not know the parameters and assume that the variance is not large, the difference between the average and the specific object is not large. Therefore, it is okay to use the specific object value instead of the average, and it can speed up the computation by not calculating the average. Since this preprocessing is inspired by the clustering algorithm, other clustering methodologies can be applied.

The big-o of the proposed algorithm is mng when there are mxn images and the addition is based on g, where g is max (the number of groups). For the same images, the big-o of box-filter with size kxk is mn\*(k^2) and the gaussian is mnE (where E is the E of of E, which is usually a very large number). In simulation, the number of clusters in the proposed algorithm is often larger than k^2, so it is more computationally intensive than box-filter, but less than gaussian. However, the existing method has a local property because it uses the neighboring information of the pixel, while the proposed preprocessing algorithm has a global property because it performs the comparison for all pixels. This avoids the bias caused by the traditional algorithm's calculation of pixel neighborhood values and allows pixels with similar RGB or brightness to be classified into the same group, which means that it is more flexible and reduces information loss.

Based on this preprocessing, the method for classifying pixels as edges first compares the pixel to the group of pixels to its left and right (or up and down). If any pixel is classified in a different group than the former, it is classified as an edge. This is susceptible to noise, so if necessary, the following methods can be used as additions to the existing algorithm Check the group of each pixel by referring to its neighborhood, and then find the ratio of the group type to the number of pixels examined. If the value is greater than a threshold, the pixel is not classified as an edge. This is similar to how existing algorithms check the orientation of the surrounding pixels when determining noise: because the neighborhood of a particular pixel is not smooth, the pixel's surroundings take on different values, resulting in different kinds of groups, but the pixel is not classified as an edge because it does not form the boundary of an object in the image. This is similar to the smoothing of a box or gaussian filter, but there is no bias because it does not replace the information of a specific pixel with the computational results of the surrounding objects.

3.Simulation

First of all, I wrote the algorithm described above in python3.8.10 through Jupyter notebook on lubuntu (a derivative OS of Ubuntu). The main specifications of the simulated computer are CPU (i3-9100), 16GB of Ram, and 120GB of SSD. <Figure 1> shows two photos used to check the performance of the algorithm, the first of which was provided as a final assignment (Sample 2). The reason for this is that the information stored in each pixel is a stochastic variable due to the nature of photons, so we want to see how well the algorithm can select edges in the presence of noise around the pixels. The second image is a scene from the game StarCraft 2 (hereafter Star2), which is characterized by similar brightness and RGB colors of the object and the background. This is a difficult image to extract edges from, and we believe that it is important to be able to extract edges in this situation, so we used it for testing.

To summarize the simulation results First of all, the lower the brightness threshold or the higher the threshold (similarity) in RGB (RGBL: a four-dimensional space that adds brightness information to the traditional RGB space), the noisier and more detailed the image. And algorithms that remove noise and obtain edges are effective in noisy situations (low thresholds in brightness or high similarity in RGB (or RGBL)). This denoising can result in good edges with high thresholds, especially in RGB or RGBL, which are even better than edges with low thresholds. However, in situations where the threshold is high and noise is removed, there is no difference between edge detection algorithms. Also, in situations where brightness is similar, data preprocessing with color information (or color and brightness) yields better edges than brightness alone, but discontinuous edges or similar brightness between neighboring pixels may not be classified as edges.

First of all, if we look at the preprocessing and edge detection results for sample 2 using brightness in Figure 2-1 and Figure 2-2, we can see that threshold and detail are inversely related and denoising is proportional. However, we can see that the results are similar when the threshold is median and when the significance level is 0.95, but if we look at the results for other data in <Figure 5-2>, we can see that this situation occurs only for sample 2. And when the threshold is above the median, the detail is very poor, and if you look at the sample 2 photo in Figure 1, you can guess that it is related to the similar brightness of the photos. That is, the performance of edge extraction based on the brightness is poor because it is similar. And the denoising effect when determining the edge is only effective in noisy situations (thres hold is 005 (see Figure 2-1)).

<Figure 3-1 and Figure 3-2 show the results of preprocessing Sample 2 with COLOR and obtaining EDGE. First of all, the overall detail is higher than when using brightness. However, pixels that were declared as edge using brightness were not declared as edge using color. We speculate that this is caused by the similarity of the sky color and roof color around the pixel. It is worth noting that in some areas, denoising edge(0.9 5) in <Figure 3-2> performs better than denoising edge(0.9). Specifically, the former does a better job of representing the yellow-colored region in sample 2 of <Figure 1> without noise than the latter.

<Figure 4-1 and Figure 4-2 show the results of using color and brightness simultaneously. It can be seen that for some areas, the edge obtained by using both color and brightness is more detailed and less noisy than the edge obtained by using color alone, assuming that the threshold is 0.95 and the noise is removed. However, in the yellow areas, the former (using color only) has less noise and more detail than the latter (using color and brightness). The reason for this difference is currently unclear and needs to be studied.

<Figure 5-1 and Figure 5-2 preprocess the Star2 image using brightness to obtain the edge. When comparing <6-2> and <7-2>, you can see that the performance is very poor. Due to the nature of the data, there are many places where the object and background are similar, so you can't get a good edge using brightness. Also, in <6-2>, we can see that the edge detector that denoised the noise obtained better edges than the edge detector that did not, regardless of the threshold. Specifically, some pixels inside the object were classified as edges when the noise was not removed, but after removing the noise, those pixels are no longer edges. Figure 7-2 shows that even if the threshold is set high when classifying edges, removing the noise can give better edges than when the threshold is low. In particular, in Figure 7-2, the edges obtained by denoising are less likely to misclassify pixels in the inner regions of the object as edges than otherwise.

In summary, no one preprocessing method and edge detection algorithm is always better than another, and algorithm performance depends on the characteristics of the data (such as brightness or color similarity). So one way to determine which preprocessing method to use and which edge detection algorithm to use is to first characterize the image by looking at the distribution of brightness and color. For example, if the distribution of brightness converges to a uniform distribution. We can conclude that it would be a bad idea to edge on brightness for that image. Therefore, it may be more effective to preprocess the data using color or a combination of color and brightness to get an edge.

<Figure 1: Sample 2 and StarCraft 2 playing StarCraft 2

|  |  |
| --- | --- |
|  |  |

<Figure 2-1: Image preprocessing (brightness) result for Sample 2

In parentheses is the threshold, where 095 is the significance level at which the difference in brightness values has a 5% chance of being greater than the mean from that distribution. Conversely, 005 is the significance level at which there is a 5% chance that a value less than the mean is drawn from that distribution. median is the median of the difference in brightness values.

|  |  |  |
| --- | --- | --- |
| Sample 2 Brightness Preprocessing (005) | Sample2 Brightness Preprocessing (median) | Sample 2 Brightness Preprocessing (095) |
|  |  |  |

<Figure 2-2: EDGE result obtained through brightness in Sample 2

|  |  |  |
| --- | --- | --- |
| Sample2 brightness edge(005) | Sample2 brightness edge(median) | Sample2 brightness edge(095) |
|  |  |  |
| Sample2 brightness edge(005): denoised | Sample2 brightness edge(median) denoised | Sample2 brightness edge(095) denoised |
|  |  |  |

<Figure 3-1: Preprocessing results obtained with color on sample 2

|  |  |  |
| --- | --- | --- |
| Preprocess Sample2 color (similarity 0.8) | Preprocess Sample2 color (similarity 0.9) | Sample2 color preprocessing (similarity 0.95) |
|  |  |  |

<Figure 3-2: EDGE result obtained by COLOR on sample 2> <Figure 3-3: EDGE result obtained by COLOR on sample 2

|  |  |  |
| --- | --- | --- |
| Sample2 colored edges (similarity 0.8) | Sample2 colored edges (similarity 0.9) | Sample2 colored edges (similarity 0.95) |
|  |  |  |
| Sample2 colored edges (similarity 0.8), denoised | Sample2 colored edges (similarity 0.9), denoised | Sample2 colored edges (similarity 0.95), denoised |
|  |  |  |

<Figure 4-1: Preprocessing results obtained with color and brightness on sample 2

|  |  |
| --- | --- |
| Sample 2 color and brightness preprocessing (similarity 0.9) | Sample2 color and brightness preprocessing (similarity 0.95) |
|  |  |

<Figure 3-2: EDGE result obtained by color and brightness on sample 2

|  |  |
| --- | --- |
| Sample2 color and brightness edges (similarity 0.9) | Sample2 color and brightness edges (similarity 0.95) |
|  |  |
| Sample2 color and brightness edges (similarity 0.9), denoised | Sample2 color and brightness edges (similarity 0.95), denoised |
|  |  |

<Figure 5-1: Image preprocessing (brightness) result for Star 2

|  |  |  |
| --- | --- | --- |
| Star2 Brightness Preprocessing (005) | Star2 brightness preprocessing (median) | Star2 Brightness Preprocessing (095) |
|  |  |  |

<Figure 5-2: EDGE result obtained through brightness on Star2

|  |  |  |
| --- | --- | --- |
| Star2 brightness edge(005) | Star2 brightness edge(median) | Star2 brightness edge(095) |
|  |  |  |
| Star2 brightness edge(005): denoised | Star2 brightness edge(median) denoised | Star2 brightness edge(095) denoised |
|  |  |  |

<Figure 6-1: Preprocessing results obtained with COLOR on STAR2

|  |  |  |
| --- | --- | --- |
| Star2 color preprocessing (similarity 0.8) | Star2 color preprocessing (similarity 0.9) | Star2 color preprocessing (similarity 0.95) |
|  |  |  |

<Figure 3-2: EDGE result obtained by COLOR on Sample 2> <Figure 3-3: EDGE result obtained by COLOR on Sample 2

|  |  |  |
| --- | --- | --- |
| Star2 colored edges (similarity 0.8) | Star2 colored edges (similarity 0.9) | Star2 colored edges (similarity 0.95) |
|  |  |  |
| Star2 colored edges (similarity 0.8), denoised | Star2 colored edges (similarity 0.9), denoised | Star2 colored edges (similarity 0.95), denoised |
|  |  |  |

<Figure 7-1: Preprocessing results obtained with color and brightness on Star 2

|  |  |
| --- | --- |
| Star2 color and brightness preprocessing (similarity 0.9) | Star2 color and brightness preprocessing (similarity 0.95) |
|  |  |

<Figure 7-2: EDGE result obtained by color and brightness in Sample 2

|  |  |
| --- | --- |
| Star2 color and brightness edges (similarity 0.9) | Star2 color and brightness edges (similarity 0.95) |
|  |  |
| Star2 color and brightness edges (similarity 0.9), denoised | Star2 color and brightness edges (similarity 0.95), denoised |
|  |  |

5.5 Limitations and Future Research

First of all, my proposed algorithm has some limitations. First of all, I proposed a statistical method to determine the threshold in brightness, but not in RGB and RGBL. Initially, I wanted to find the similarity of all pixels and then select a p-value with a statistical significance level of 5%, similar to how I found the threshold for brightness. However, this was not implemented because it was too computationally intensive, and I did not try the method of selecting some data by sampling and determining the threshold because of the possibility of bias in the sample. Also, even if the threshold is determined statistically, the problem of using only one threshold is still present: too high a threshold results in a loss of detail, while too low a threshold results in too much noise.

To address this, we currently check if a pixel is noise when classifying an edge, but an alternative approach is to combine the results of using more than one threshold in an appropriate way. We can combine edge results from different preprocessing methods but with different thresholds, but we can also combine edge results from different preprocessing methods. The idea is to combine the best of each preprocessing method, as shown in the simulation. In addition, classifying pixels in RGB or RGBL can be computationally intensive, which can be overcome by using threads or modifying the algorithm to be matrix-based to perform the computation on the graphics card. In general, the proposed algorithms sometimes obtain edges discontinuously, so we need to think about how to obtain edges continuously while reducing noise. Also, in the future, we would like to use differential geometry methodologies to process data and obtain edges more effectively. Finally, we plan to perform image preprocessing using other clustering algorithms.